Developer Advisory

"Client-side Form Validation Using JavaScript"

Issued Date: ..... September 21, 2001
Submitted by: ..... Joe Nelson

Overview

When you have a form on a web page, you'll most likely find the need to validate the values a user enters. Client-side validation has advantages for the server, the user and the developer.

Validating forms on the client side lightens the load on the server and allows for you to format data before it is sent for processing. Because validation takes place on the client machine, there's no delay from the client contacting a remote server. Besides the user getting a quicker response, network bandwidth and server processing power are both conserved.

Using server-side validation, there is noticeable lag time when data the user submits contains an error. Form data must travel over the Internet and be examined by the server. Eventually, the user will receive a response page only to discover that there was an error and that the submittal process needs to be repeated. Running a little preliminary validation of the user's form input before it is submitted eliminates the wasted time.

Client-side validation, conversely, is instantaneous because it doesn't have to transmit any data. JavaScript catches any erroneous data the user enters before it goes anywhere.

Developers will find it easier and more intuitive to do form validation with JavaScript than with server side techniques since the code has already been written and is easy to implement. It also makes the task of dealing with the form information a lot easier since the data is pre-formatted before being sent to the server. Client-side form validation is easy, quick, efficient, and user-friendly.

Continued …

JavaScript Approach

We've taken a modular approach and created a collection of discrete JavaScript functions that can be used to validate just about any type of form field. The entire code can be placed or referenced in your web page or you may use just the functions that apply to the field types you are using.

This approach uses:

- A global error array that can be used for error trapping and reporting.

- Form fields that automatically reformat so users can enter their information in various ways and the JavaScript will be able to interpret it

- JavaScript's regular expressions, which are a notation for describing sequences of characters, to simplify and reduce the size of the code.

DHML to create and populate some form fields on the fly.

For example:

The function for creating date drop down lists that automatically updates itself with the correct number of days in the currently chosen month, taking into account things like leap years.

Continued …

Instructions for use

Overview:

This Javascript form validation library is a collection of functions that you can use to validate just about any type of form field available, or to actually create some of the standard fields. To take advantage of this library, follow these three steps:

1. Include the javascript library in your page.
2. Create a validation function to check each field you want inspected.
3. Add the "onClick()" event handler to a button or link that will submit the form.

Details:

1. Include the javascript library in your page.

   Place the following inside the `<head>` and `</head>` tags:

   ```
   <script language="JavaScript" src="formcheck.js"></script>
   ```

2. Create a validation function to check each field you want inspected.

   Create a function called "validate" (Refer to the example page). This function should check each required field. The field checks will automatically populate an error array if there is a problem with the field's value. After checking each field, see if there are any errors. If so, let the user know by opening an error window. If not, trigger whatever action is appropriate upon successful completion of the form. This can be redirecting to another page, submitting the form, etc. Use the function reference below when creating the validation function.

Continued …

3. Add the "`onClick()`" event handler to a button or link that will submit the form.

   To submit the form that will be validated, use a "`button`" form element instead of a "`submit`" element such as the following:

```
<input type="button" name="val" value="Submit"
onClick="validate(this.form)">
```

   Be sure to call the validate function on the button's onClick event handler as shown above.

Continued …

Function Reference:

These functions can be used anywhere, throughout your page. Field validation and error checking are most often used when validating an entire form before submitting. Field creation can be used wherever you want the field to appear. Some functions have an optional "emptyOK" parameter. You can set this to true if you don't want the field to be required, but still want it checked for accuracy if filled in or if you just want it reformatted before sending it to the server. This can be used for things like a fax field (See example.html).

Field Validation

checkString(theField, error)

- Check that theField.value is not empty or all whitespace.
- theField = Form field to validate
- error = Error string to return if unsuccessful

checkStateCode(theField [, error])

- Check that theField.value is a valid U.S. state code.
- theField = Form field to validate
- error = Error string to return if unsuccessful - defaults to eState

checkZIPCode(theField [, error, emptyOK])

- Check that theField.value is a valid ZIP code.
- theField = Form field to validate
- error = Error string to return if unsuccessful - defaults to eZip
- emptyOK = Can this field be empty?

Continued …

checkUSPhone(theField [, error, emptyOK])

- Check that theField.value is a valid US Phone.
- theField = Form field to validate
- error = Error string to return if unsuccessful - defaults to ePhone
- emptyOK = Can this field be empty?

checkInternationalPhone(theField [, error, emptyOK])

- Check that theField.value is a valid International Phone.
- theField = Form field to validate
- error = Error string to return if unsuccessful - defaults to eIntPhone
- emptyOK = Can this field be empty?

checkEmail(theField [, error])

- Check that theField.value is a valid Email.
- theField = Form field to validate
- error = Error string to return if unsuccessful - defaults to eEMail

checkSSN(theField [, error, emptyOK])

- Check that theField.value is a valid SSN.
- theField = Form field to validate
- error = Error string to return if unsuccessful - defaults to eSSN
- emptyOK = Can this field be empty?

Continued …

checkDate(yearField, monthField, dayField, error [, OKtoOmitDay])

- Check that field values form a valid date.
- yearField = Form field that contains the year
- monthField = Form field that contains the month
- dayField = Form field that contains the day
- error = Error string to return if unsuccessful
- OKtoOmitDay = Does this form require a day to be entered? (ie credit cards do not)

checkCreditCard(typeField, theField [, error])

- Validate credit card info.
- typeField = Form field that contains the credit card type
- theField = Form field the contains the credit card number to be checked
- error = Error string to return if unsuccessful - defaults to eCCNumber

Field Creation

writeStates([selected])

- Creates a US states drop down list. Field name is "state"
- selected = Initially selected element - defaults to UT

writeCounties([selected])

- Creates a Utah counties drop down list. Field name is "counties"
- selected = Initially selected element - defaults to Salt Lake

writeCC(name [, selected])

- Creates a credit card type drop down list. Field name is the name you pass to the function
- name = Name to give the form field that will be created
- selected = Initially selected element - defaults to Visa

Continued …

writeMonths(group [, selected])

- Creates a months drop down list. Field name is "months_" + group
- group = Group this field belongs to
- selected = Initially selected element - defaults to this month

writeDays(group [, noday, selected])

- Creates a days drop down list. This will be dynamically updated based on the currently selected month and year in this group. (See the "group" parameter.) Field name is "days_" + group
- group = Group this field belongs to
- selected = Initially selected element - defaults to today

writeYears(group, start, end [, selected])

- Creates a years drop down list. Field name is "years_" + group
- group = Group this field belongs to
- start = Year to start with
- end = Year to end with
- selected = Initially selected element - defaults to this year

writeSalutations([selected])

- Creates a drop down box with common salutations. (Mr, Mrs, etc). Field name is "salutation"
- selected = Initially selected element - defaults to none ("")

writeSuffixes([selected])

- Creates a drop down box with common suffixes. (Jr, Sr, etc). Field name is "salutation"
- selected = Initially selected element - defaults to none ("")

Continued …

Error Handling

returnError([newLine])

- Return the error string created during the form field validations.
- newLine = Charater to use as a new line character - defaults to '
'

hasErrors()

- Does this form have any errors?

errorWindow([error, width, height, startHTML, endHTML])

- Creates a centered popup window that tells the users where the errors in the from are.
- error = Error string to return if unsuccessful - defaults to returnError()
- width = Width of the window in pixels - defaults to 300
- height = Height of the window in pixels - defaults to 300
- startHTML = Any HTML or text to be put before the error is printed - defaults to ""
- endHTML = Any HTML or text to be put after the error is printed - defaults to ""